

Supplementary Materials for Manuscript entitled: “*Computational design of syntheses leading to compound libraries or isotopically labelled targets.*” by Karol Molga^{1‡}, Piotr Dittwald^{1‡}, Bartosz A. Grzybowski^{1,2*}

¹ Institute of Organic Chemistry, Polish Academy of Sciences, ul. Kasprzaka 44/52, Warsaw 01-224, Poland

² IBS Center for Soft and Living Matter and Department of Chemistry, UNIST, 50, UNIST-gil, Eonyang-eup, Ulju-gun, Ulsan, 689-798, South Korea

[‡] Authors contributed equally

*Correspondence to: nanogrzybowski@gmail.com

CONTENTS:

Section S1. Pseudocode of the algorithm seeking the syntheses of all targets.

Section S2. Pseudocode of the algorithm seeking the easiest syntheses of some targets.

Section S3. Pseudocode of the algorithm generating isotopomers with a desired mass shift.

Section S4. Comparison of Chematica’s searches to make all members of a library vs. consecutive searches for individual targets.

Section S5. Details of Chematica’s retrosynthetic analyses performed for fluoxetine derivatives.

Section S6. Details of Chematica’s retrosynthetic analyses performed for fluoxetine derivatives in individual, target-by-target searches.

Section S7. Details of Chematica’s retrosynthetic analyses performed for Almorexant derivatives.

Section S8. Details of Chematica’s retrosynthetic analyses performed for tryptophan derivatives.

Section S9. Details of Chematica’s retrosynthetic analyses performed for the ICI199441 derivatives.

Section S10. Details of Chematica’s retrosynthetic analyses performed for ¹³C/²H labeled Cinacalcet.

Section S11. Details of Chematica’s retrosynthetic analyses performed for various M+1 ¹³C labelled drug molecules.

Section S1. Pseudocode of the algorithm seeking the syntheses of all targets.

```
1: function GENERATEDUMMYREACTIONAND(TS)
  ▷ TS: targets set (library) composed of  $\{target_1, target_2, \dots, target_N\}$ 
  ▷ function generates and returns dummy reaction where TS is a product,
    and  $target_1, \dots, target_N$  are substrates
  ▷ therefore, to find retrosynthetic path for TS, search algorithm needs to
    find viable retrosynthetic scenarios for  $target_1$  AND ... AND  $target_N$ 
2:   r ← newReaction()
3:   r.addProduct(TS)
4:   for target in TS do
5:     r.addSubstrate(target)
   return {r}

6: procedure SEARCHFORLIBRARYAND(TS)
  ▷ TS: targets set (library) composed of  $\{target_1, target_2, \dots, target_N\}$ 
7:   put(PQ, node( $\{TS\}$ ))
8:   dummyReactionNotYetGenerated ← True
9:   initializeSearchGraph(TS)
10:  while True do
11:    substratesNode ← pop(PQ)
12:    for substrate in getSubstrates(substratesNode) do
13:      if dummyReactionNotYetGenerated then
14:        progenySet ← generateDummyReactionAND(substrate)
15:        dummyReactionNotYetGenerated ← False
16:      else
17:        progenySet ← generateRetrosynthesisSteps(substrate)
18:        for progeny in progenySet do
19:          addToSearchGraph(substrate, progeny)
20:          newSubstratesNode ← node((getSubstrates(substratesNode) –
            {substrate}) ∪ progeny)
21:          if notEndOfSynthesis(newSubstratesNode) then
22:            put(PQ, newSubstratesNode)
```

Figure S1. The algorithm seeking the syntheses of all targets is the extension of existing routines for retrosynthesis search. Here, we present the pseudo-code for such a search procedure

(*searchForLibraryAND*, lines 6-22), that is run for target set, *TS*, being a user-defined library of targets $\{target_1, ..., target_N\}$. The algorithm puts a node for the target set $\{TS\}$ into priority-queue-based data structure, *PQ*, analogous to the one used for single-target search. Further, the algorithm initializes *dummyReactionNotYetGenerated* as *True*, and search graph with a single chemical node representing $\{TS\}$ (lines 7-9). Then, the while-loop begins (the loop might be terminated, e.g., when the user decides to stop the search, if satisfactory pathways are found, or after a defined number of iterations are performed – here, loop termination is not explicitly considered for code-brevity reasons. In the first iteration of the loop, (*dummyReactionNotYetGenerated* is *True*), the algorithm calls *generateDummyReactionAND* (line 14), returning *progenySet* composed of the “multicomponent” dummy reaction $target_1, ..., target_N \rightarrow TS$ (lines 1-5), which is further added to the search graph (line 19). Subsequently, $newSubstratesNode = node(\{target_1, ..., target_N\})$ is put to *PQ* (line 22), which prioritizes the constituent nodes according to the user-provided scoring functions (see main text). As variable *dummyReactionNotYetGenerated* has been set to *False* (line 19), and will not be changed anymore, in all the subsequent iterations of the while-loop, the *progenySet* is computed as a collection of viable retrosynthetic steps (*generateRetrosynthesisSteps*, line 21). In Chematica, the related computations are based on expert-coded reaction rules, and include detection of possible cross-reactivity conflicts, protections, non-selectivity issues, etc. (see main text for references). As new retrosynthetic steps are generated, the search graph is expanded (line 19), and new synthetic options are added to *PQ* (line 22). The separate selection algorithm (see main text) is applied to retrieve a diverse set of viable retrosynthetic solutions for the requested library of targets. Please note that in the presented pseudo-code, implementation-specific optimizations such as code parallelization or search-graph representation are not considered.

Section S2. Pseudocode of the algorithm seeking the easiest syntheses of some targets.

```

1: function GENERATEDUMMYREACTIONOR(TS)
  ▷ TS: targets set (library) composed of  $\{target_1, target_2, \dots, target_N\}$ 
2:   progenySet  $\leftarrow \emptyset$ 
3:   for target in TS do
4:     r  $\leftarrow$  newReaction()
5:     r.addProduct(TS)
6:     r.addSubstrate(target)
7:     progenySet.update(r)
  return progenySet

8: function EXTENDIFNEEDED(PL, substratesToBeAnalyzed)
  ▷ PL: priority list; substratesToBeAnalyzed: list with substrate nodes
9:   while extensionNeeded do
10:    if allQueuesEmpty(PL) then raise Exception('PL empty')
11:    if notEmpty(PL[PL.recentlyVisited]) then
12:      substratesNode = pop(PL[PL.recentlyVisited])
13:      substratesNode.indexTakenFrom = PL.recentlyVisited
14:      substratesToBeAnalyzed.extend(substratesNode)
15:      PL.recentlyVisited  $\leftarrow$  (PL.recentlyVisited + 1) % PL.size

16: procedure SEARCHFORLIBRARYOR(TS)
  ▷ TS: targets set (library) composed of  $\{target_1, target_2, \dots, target_N\}$ 
  ▷ PQ: already initialized priority-queue based data structure used in single-
  target search
17:   substratesToBeAnalyzed  $\leftarrow$  [node(TS)]
18:   dummyReactionNotYetGenerated  $\leftarrow$  True
19:   initializeSearchGraph(TS)
20:   PL  $\leftarrow$  initializePL(PQ, N)
21:   while True do
22:     substratesNode  $\leftarrow$  getNew(substratesToBeAnalyzed)
23:     for substrate in getSubstrates(substratesNode) do
24:       if dummyReactionNotYetGenerated then
25:         progenySet  $\leftarrow$  generateDummyReactionOR(substrate)
26:         dummyReactionNotYetGenerated  $\leftarrow$  False
27:         index  $\leftarrow$  0
28:         for progeny in progenySet do
29:           addToSearchGraph(substrate, progeny)
30:           putToPL(PL[index], node((getSubstrates(substratesNode) –
31:             {substrate})  $\cup$  progeny))
32:           index += 1
33:         else
34:           progenySet  $\leftarrow$  generateRetrosynthesisSteps(substrate)
35:           for progeny in progenySet do
36:             addToSearchGraph(substrate, progeny)
37:             newSubstratesNode  $\leftarrow$  node((getSubstrates(substratesNode) –
38:               {substrate})  $\cup$  progeny)
39:             if notEndOfSynthesis(newSubstratesNode) then
40:               putToPL(PL[substratesNode.indexTakenFrom], newSubstratesNode)
41:             extendIfNeeded(PL, substratesToBeAnalyzed)

```

Figure S2. The algorithm seeking the easiest syntheses of some targets extends routines used for single-target retrosynthesis search. As presented in the pseudo-code, the search procedure (*searchForLibraryOR*, lines 16-39) is run for the target set, $\{TS\}$, i.e., a user-defined library of N targets $\{target_1, \dots, target_N\}$. The algorithm first initializes *substratesToBeAnalyzed* as a list containing the only node for set $\{TS\}$, *dummyReactionNotYetGenerated* as *True*, and search graph with single chemical node representing $\{TS\}$ (lines 17-19). Then, the priority list *PL* is initialized (line 20) as a list of N copies of queue-based data structures, *PQ*, used in the standard single-target search. Finally, the while-loop begins (loop termination, e.g., by the user, is not explicitly considered for code-brevity reasons). As the first iteration of the loop begins, *substrate* variable is set to *node($\{TS\}$)*. The algorithm calls function *generateDummyReactionOR* (line 25), returning *progenySet* composed of N dummy reactions: $target_1 \rightarrow TS$, ..., $target_N \rightarrow TS$ (lines 1-7), which are further added to the search graph (line 29). Moreover, progenies *node($\{target_1\}$)*, ..., *node($\{target_N\}$)* are added to the subsequent elements of *PL* (line 30), allowing for comparable exploration of retrosynthetic options for each element of the initial target library. In all subsequent iterations of the while-loop, the *progenySet* is computed as a set of viable single retrosynthesis steps (line 33). In Chematica, these computations are based on expert-coded rules, and include detection of possible cross-reactivity conflicts, protections, non-selectivity issues, etc. (see main text for references). As the substrates are iteratively queried by retrosynthetic-step generator, the search graph is expanded (line 35), and *newSubstrateNode* variables corresponding to already discovered new synthetic options are added to the same element of *PL* as *substrateNode* was taken from (line 36). As algorithm advances, the list *substratesToBeAnalyzed* is extended (line 39) by taking elements from *PL*, which is inspected in a circular order (*extendIfNeeded*, line 8-15). Please note that in the presented pseudo-code, implementation-specific optimizations such as code parallelization or search-graph representation are not considered.

Section S3. Pseudocode of the algorithm generating isotopomers with a desired mass shift.

```
1: function GENERATELABELLINGS(mol, i, Scurr, S)
  ▷ mol: considered molecule
  ▷ i: index of currently analyzed atom
  ▷ Scurr: current mass shift, i.e. after processing atoms 1, ..., (i - 1)
  ▷ S: desired mass shift of the whole molecule
2:   if i > mol.NumberOfAtoms() then //all atoms already processed
3:     if Scurr = S then return canonicSmiles(mol)
     return NULL //wrong mass shift
4:   a ← mol.getAtom(i)
5:   for iso ∈ allowedIsotopes(a) do
6:     if iso.massShift() + Scurr > S then continue
7:     mol2 ← mol.copy()
8:     setIsotope(mol2, i, iso) //set i-th atom as iso
9:     yield generateLabellings(mol, i + 1, iso.massShift() + Scurr, S)

10: function GETISOTOPOMERS(mol, S)
  ▷ mol: considered molecule
  ▷ S: desired mass shift of the whole molecule
11:   V ← ∅
12:   for l ∈ generateLabellings(mol, 1, 0, S) do
13:     if l = NULL then continue
14:     V.add(l)
15:   return V
```

Figure S3. The algorithm generates plausible isotopomers (GetIsotopomers, lines 10-15) by recursively-defined analysis of possible atomic labellings (GenerateLabellings, lines 1-9). The input molecule is processed atom-by-atom, adding combinations of isotopic variants to these atoms as long as the user-defined mass shift *S* is not exceeded (lines 5-9). When all atoms are analyzed (and some of them are isotopically labelled), the currently considered isotopomer is returned if its total mass shift equals *S*, otherwise it is rejected (line 3). As the generated isotopomers are added to the final set of results (line 14), the duplicated (i.e., having the same canonical SMILES representation) entries are considered only once.

Section S4. Comparison of the searches to make all members of a library vs. consecutive searches for individual targets.

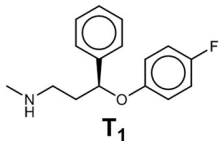
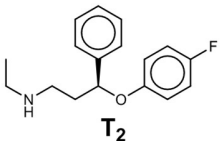
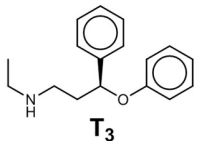
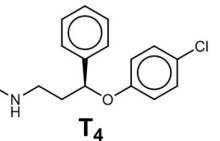
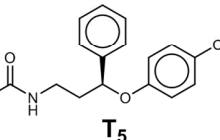
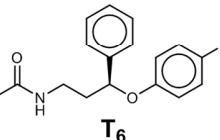
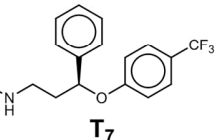
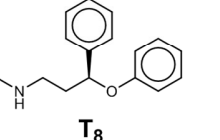
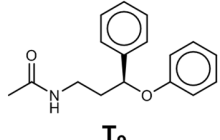
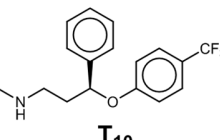
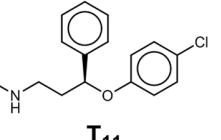
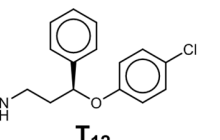
	T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	T ₇	T ₈	T ₉	T ₁₀	T ₁₁	T ₁₂	$\sum T_{1-12}$	Find-All	$\frac{\sum T_{1-12}}{\text{Find-All}}$
run1															
All nodes	3703	3653	3572	1314	1095	1924	5616	3138	1816	6577	3818	3584	39810	4228	9.4
Chemical	1553	1534	1494	569	490	802	2219	1315	764	2604	1550	1463	16357	1653	9.9
Expanded	76	70	74	20	16	32	92	71	36	111	77	80	755	72	10.5
Time (sec.)	20	18	21	12	22	11	61	19	10	68	26	26	314	55	5.7
Expanded/time	3.80	3.89	3.52	1.67	0.73	2.91	1.51	3.74	3.60	1.63	2.96	3.08	2.40	1.31	1.8
run2															
All nodes	4471	3721	3722	1627	1095	2486	5129	3634	1633	6569	3818	3816	41721	4228	9.9
Chemical	1872	1558	1547	699	490	1035	2038	1509	695	2594	1550	1566	17153	1653	10.4
Expanded	85	71	76	27	16	45	86	78	30	111	77	81	783	72	10.9
Time (sec.)	23	19	19	12	21	12	59	20	9	67	25	27	313	57	5.5
Expanded/time	3.70	3.74	4.00	2.25	0.76	3.75	1.46	3.90	3.33	1.66	3.08	3	2.50	1.26	2.0
run3															
All nodes	4319	3556	3297	1438	1095	2240	5802	4183	1629	6577	3690	3844	41670	4228	9.9
Chemical	1816	1499	1381	621	490	946	2287	1721	681	2604	1497	1579	17122	1653	10.4
Expanded	82	68	69	24	16	36	95	88	33	111	73	82	777	72	10.8
Time (sec.)	22	19	18	12	21	10	62	30	9	68	25	26	322	56	5.8
Expanded/time	3.73	3.58	3.83	2.00	0.76	3.60	1.53	2.93	3.67	1.63	2.92	3.15	2.41	1.29	1.9
<div> <div>  <p>T₁</p> </div> <div>  <p>T₂</p> </div> <div>  <p>T₃</p> </div> <div>  <p>T₄</p> </div> <div>  <p>T₅</p> </div> <div>  <p>T₆</p> </div> <div>  <p>T₇</p> </div> <div>  <p>T₈</p> </div> <div>  <p>T₉</p> </div> <div>  <p>T₁₀</p> </div> <div>  <p>T₁₁</p> </div> <div>  <p>T₁₂</p> </div> </div>															

Table S1: Individual measures of graph size and time elapsed for 12 single-target searches and the corresponding find-all algorithm. The searches are for the Prozac example from the main-text **Figures 4** and **5** (Markush structure C([*:1])C[C@H](Oc1ccc([*:2])cc1)c1ccccc1 with *substitutions_dct* = {1: ['N(C)', 'N(CC)', 'N(C(=O)C)'], 2: ['H', 'F', 'Cl', 'C(F)(F)(F)']}). The individual targets are shown under the Table.

All searches were performed on a machine with 64-processor threads clocked @2.2-3.6 GHz each. For each run, as soon as the target molecule (or all target molecules in case of the find-all algorithm) became synthesizable, the timings and the search graphs were saved. Then, the following parameters were computed for each saved graph: number of all nodes, number of chemical-substance nodes (i.e., circular nodes representing chemicals), number of chemical nodes that were expanded (i.e., retrosynthetic options for related chemicals were already computed and added to the graph). An additional parameter – ratio of expanded nodes to time – was added to estimate search efficiency. The measurements were repeated three times (run1, run2, run3). The find-all algorithm performed around 5 times faster and required around 10 times fewer nodes than 12 consecutive, single-target searches. Interestingly, we observe that find-all search needs less expanded nodes than certain individual searches (e.g., for targets T₁, T₇ or T₁₂), which might reflect ‘synergy’ between targets in the find-all mode (i.e., retrosynthetic steps explored for synthesis of one individual target might be utilized in synthetic pathways of other targets). Of note, the searches with larger number of expanded nodes tend to have higher node expansion ratio plateauing around 3-4 nodes/second. However, this is not surprising, as we generally anticipate fewer but more complex molecules (e.g. late intermediates) to be analyzed at the beginning of the search vs. larger numbers of simpler molecules to be analyzed later in the search. Additionally, we note that searches performed for trifluoromethylated targets (T₅, T₇ and T₁₀) have lower ratios of node expansion (**Chart S1**) compared with other library members. This can be explained by our observation that execution of retrosynthetic steps involving highly symmetric groups, such as CF₃, is relatively more computationally demanding, possibly due to multiple transformation-to-molecule matchings.

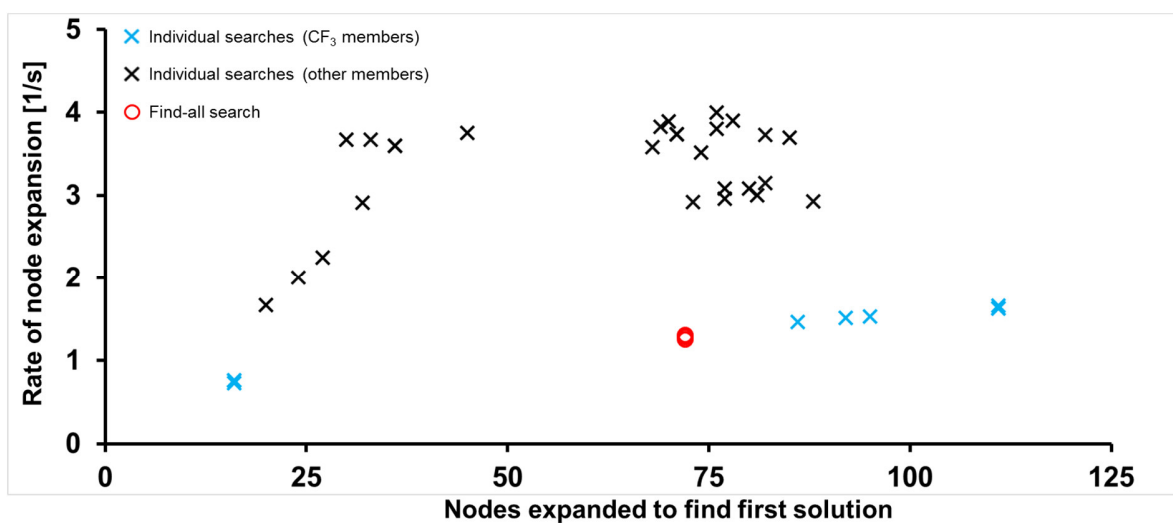


Chart S1. Rates of node expansion (i.e., number of expanded nodes per unit time) during single target and find-all searches. Blue crosses denote observed node expansion rates for trifluoromethylated library members T₅, T₇ and T₁₀. Node expansion rates for other library members and find-all search are denoted by black crosses and red circles, respectively.

Section S5. Details of Chematica's retrosynthetic analyses performed for fluoxetine derivatives.



Figure S4. Details of Chematica's synthetic plan for the library of fluoxetine derivatives; the figure complements main-text **Figure 4**.

Section S6. Details of Chematica's retrosynthetic analyses performed for fluoxetine derivatives in individual, target-by-target searches.

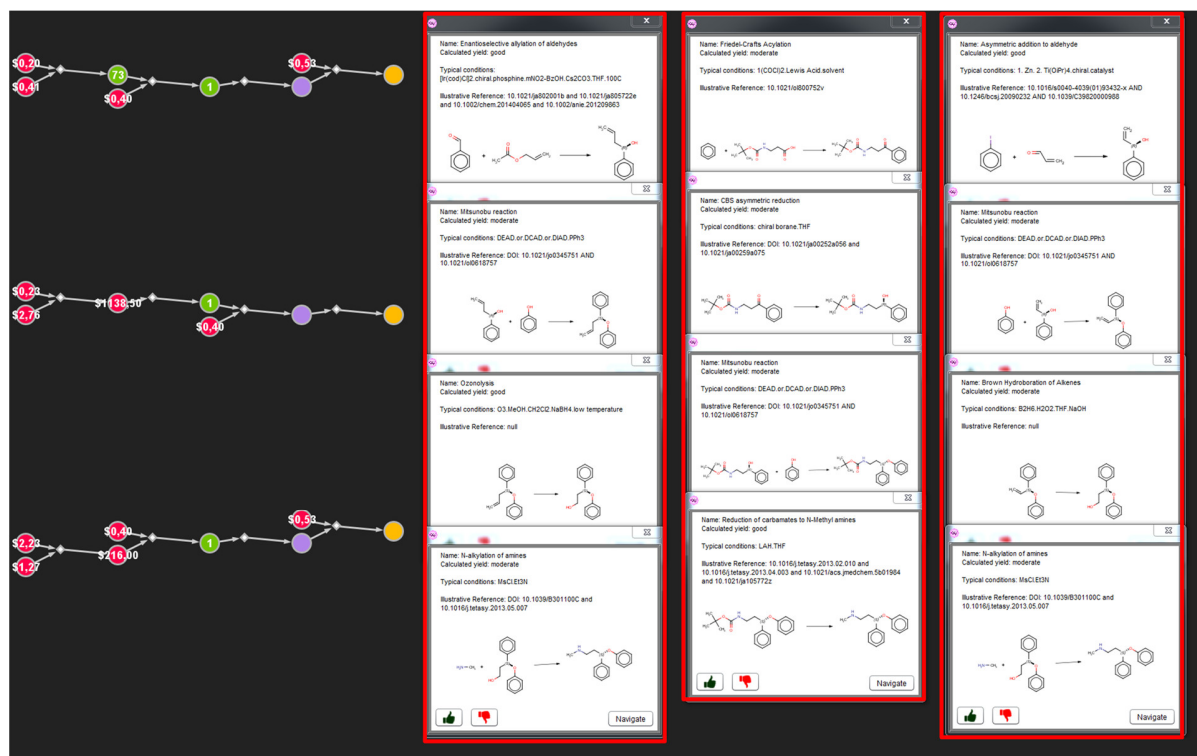


Figure S5. Details of the top-scoring pathways identified by Chematica for the A1 member of the library of fluoxetine derivatives (cf. main-text **Figure 5a-c**).

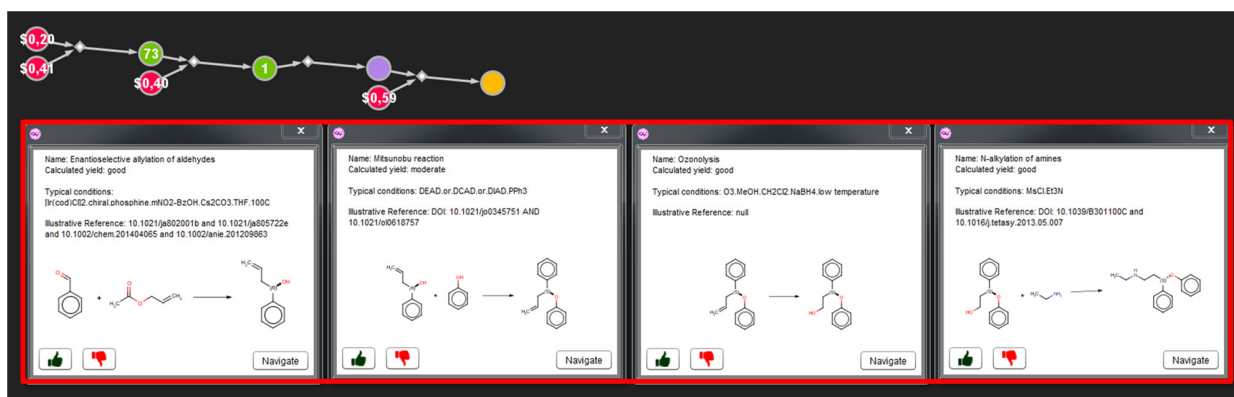


Figure S6. Details of the top-scoring pathway identified by Chematica for the **A3** member of the library of fluoxetine derivatives (cf. main-text **Figure 5d**).

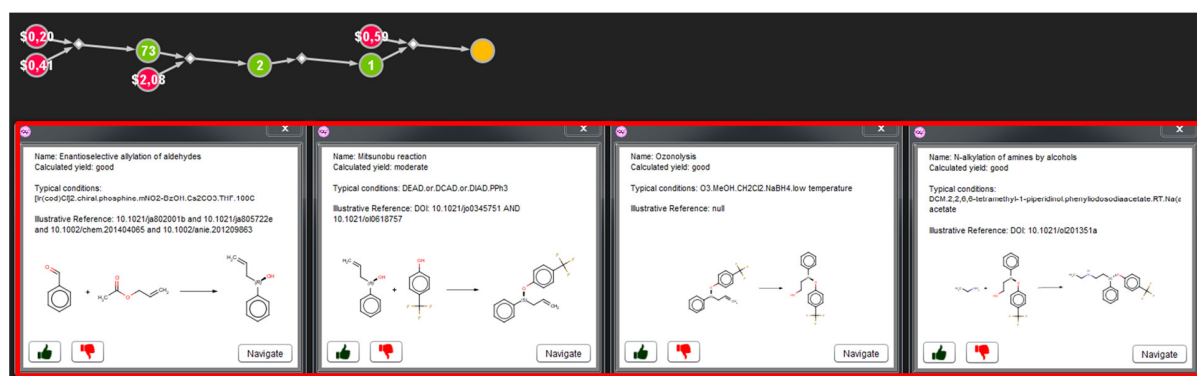


Figure S7. Details of the top-scoring pathway identified by Chematica for the **D3** member of the library of fluoxetine derivatives (cf. main-text **Figure 5e**).

Section S7. Details of Chematica's retrosynthetic analyses performed for Almorexant derivatives.

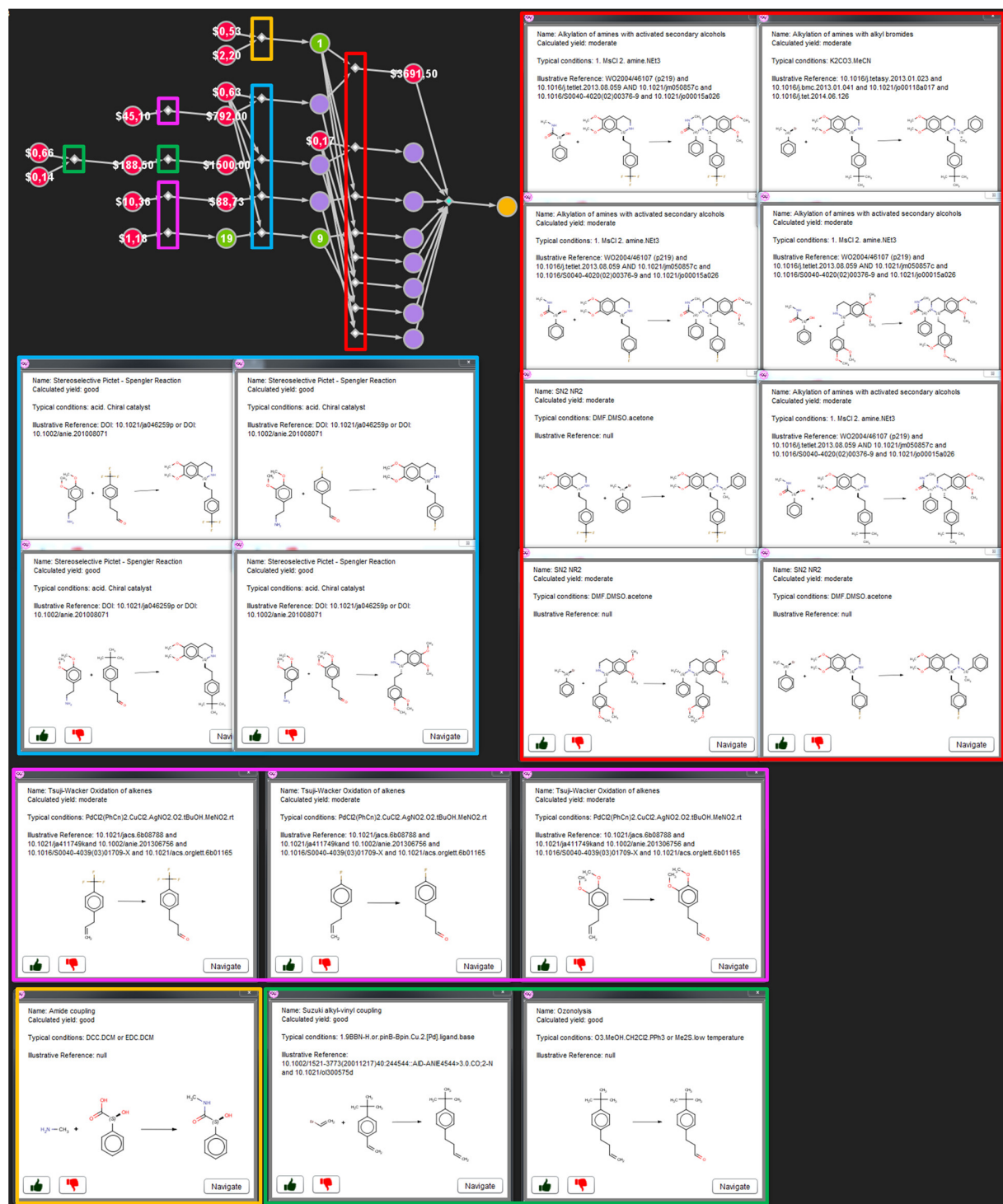


Figure S8. Details of Chematica's synthetic plan for the library of Almorexant derivatives; the figure complements main-text Figure 6.

Section S8. Details of Chematica's retrosynthetic analyses performed for tryptophan derivatives.

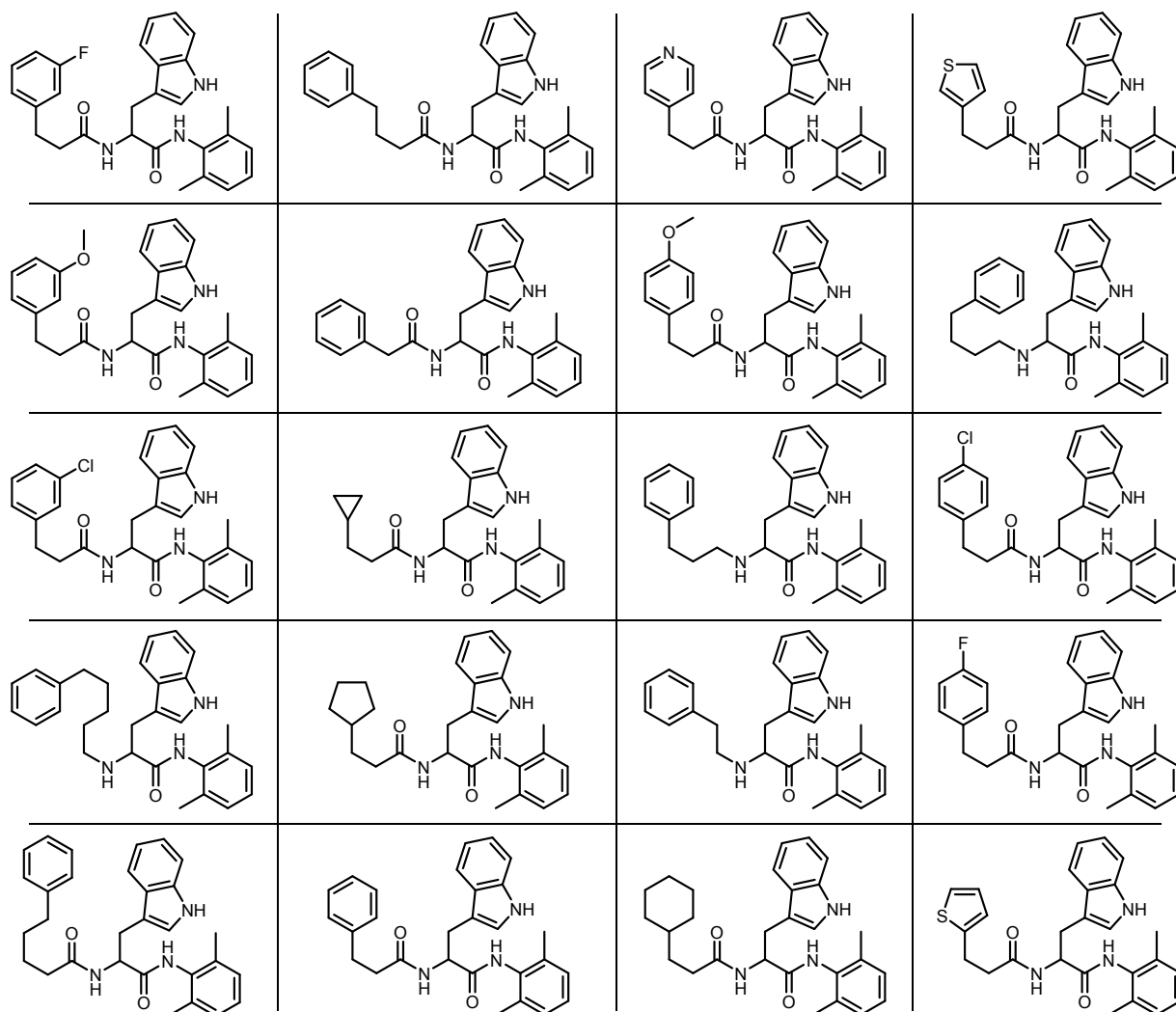


Figure S9. Components of library of tryptophan derivatives; the figure complements main-text **Figure 7**.

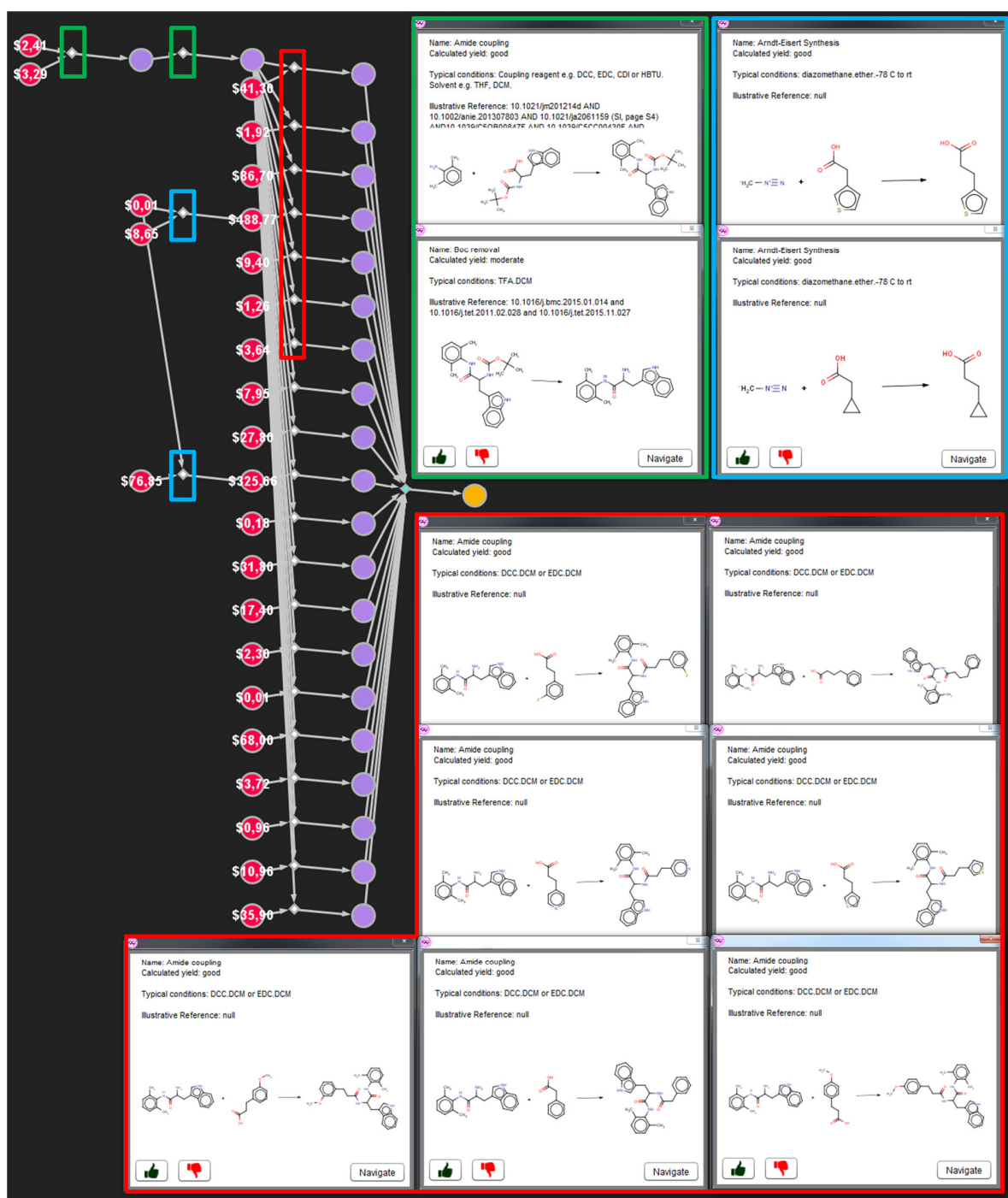


Figure S10. Details of Chematica's synthetic plan for the library of tryptophan derivatives; the figure complements main-text **Figure 7**.

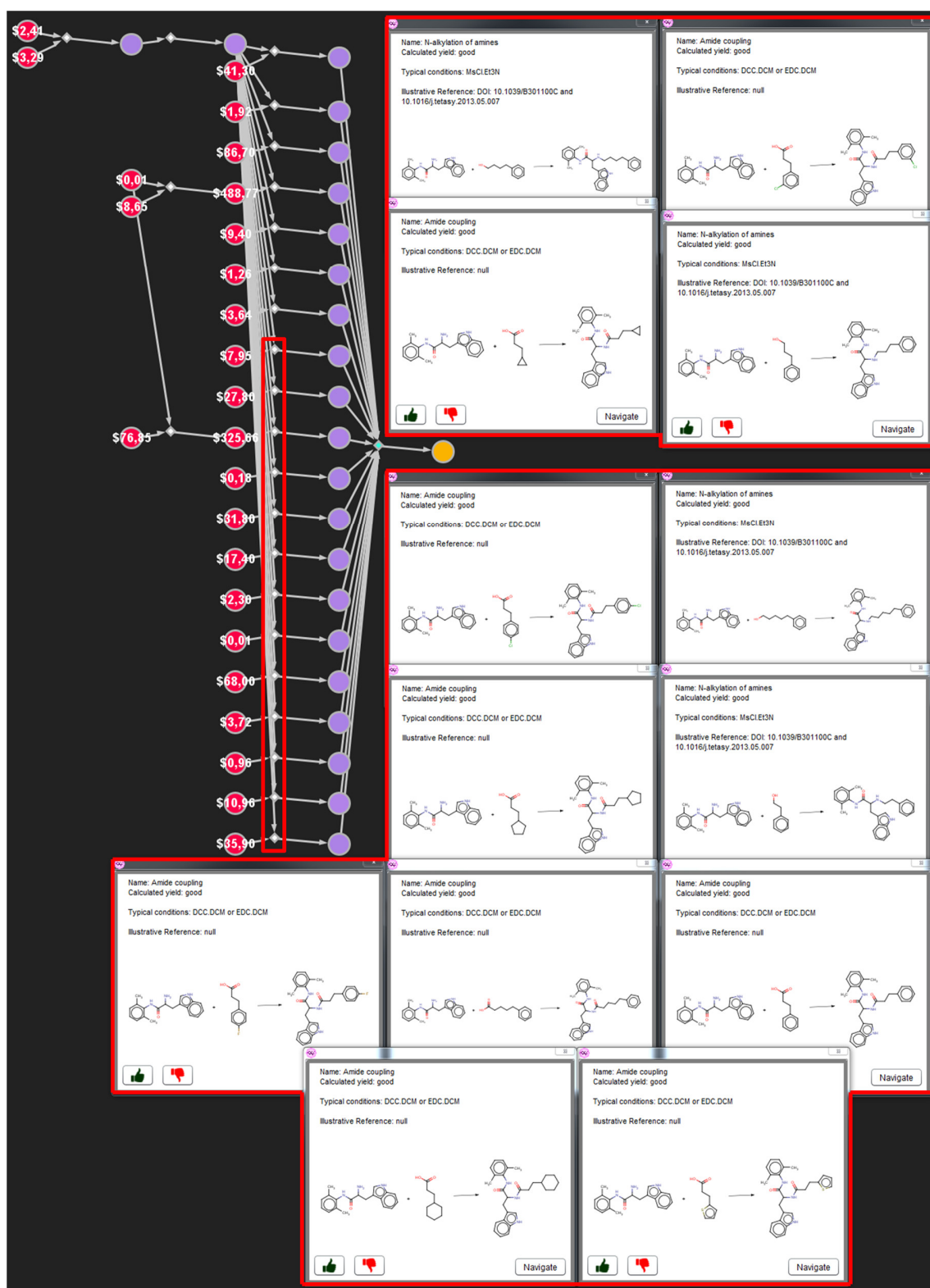


Figure S11. Details of Chematica's synthetic plan for the library of tryptophan derivatives; the figure complements main-text **Figure 7**.

Section S9. Details of Chematica's retrosynthetic analyses performed for the ICI199441 derivatives.

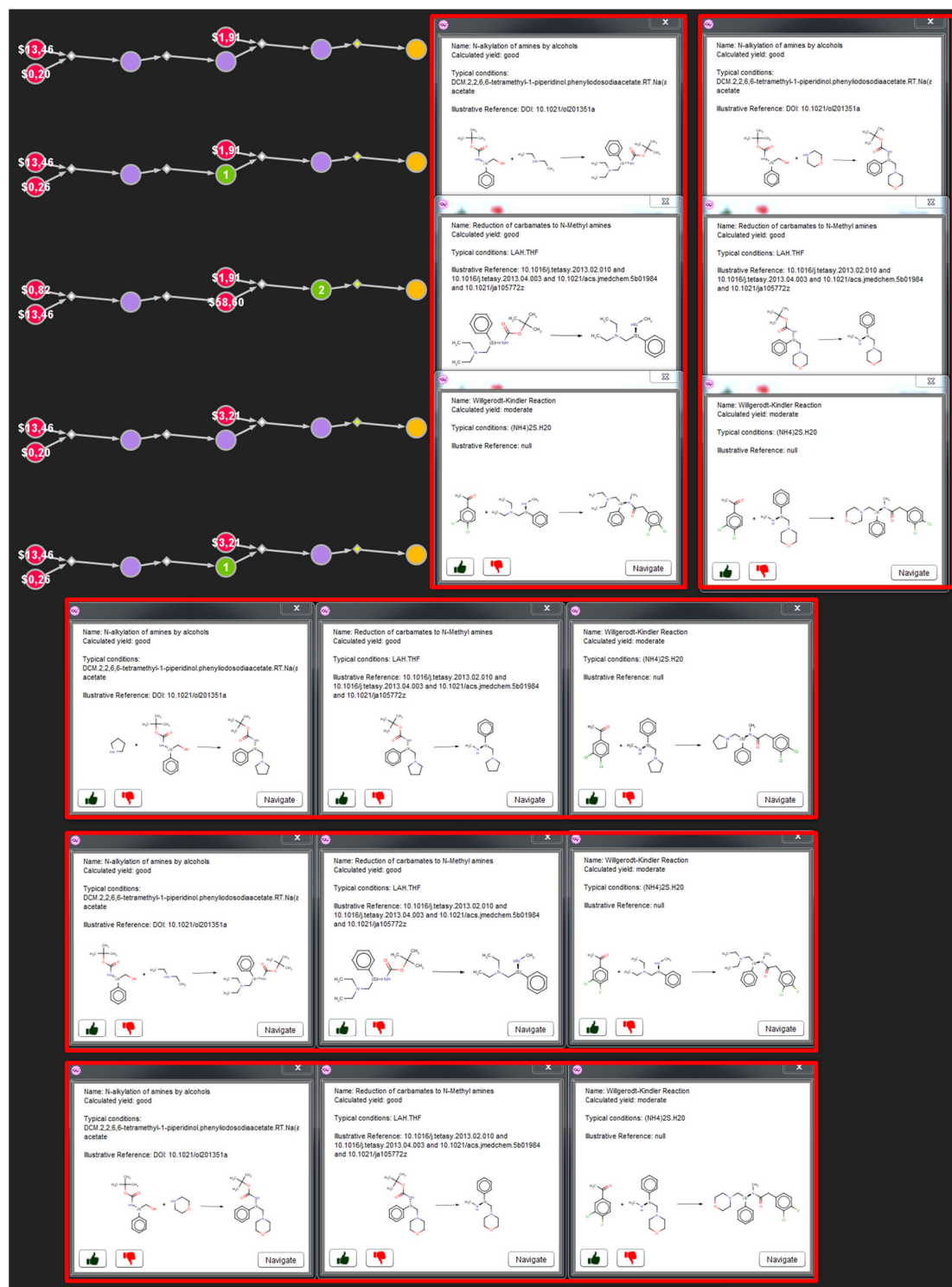


Figure S12. Details of the five top-scoring pathways proposed by Chematica's for the synthesis of the most accessible members of the ICI199441 library discussed in main-text **Figure 8**.

Section S10. Details of Chematica's retrosynthetic analyses performed for $^{13}\text{C}/^2\text{H}$ labeled Cinacalcet.

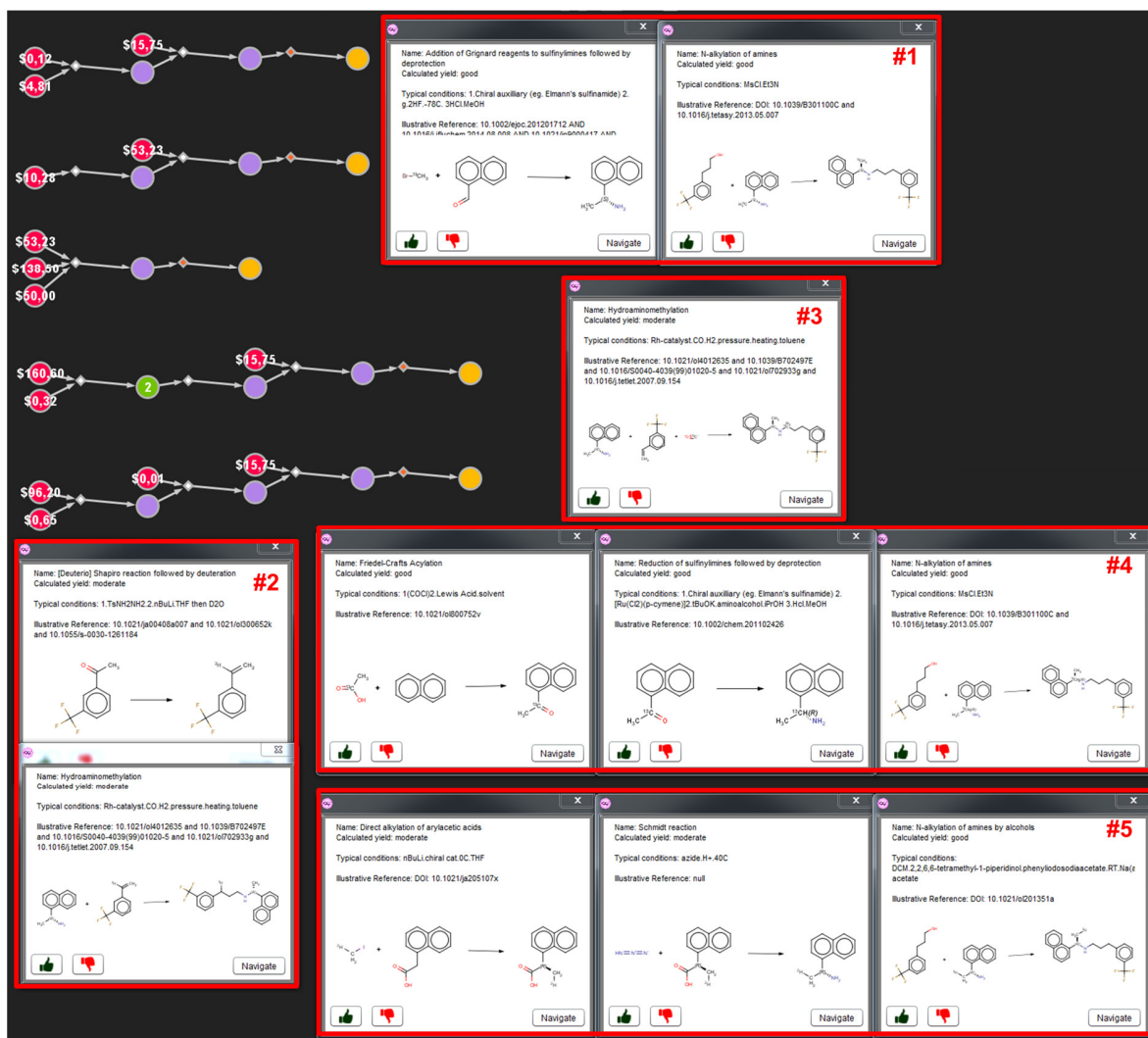


Figure S13. Details of the five top-scoring pathways identified by Chematica searching for the most accessible $^{13}\text{C}/^2\text{H}$ labeled M+1 isotopomers of Cinacalcet (cf. main-text Figure 9).

Section S11. Details of Chematica's retrosynthetic analyses performed for various M+1 ^{13}C labelled drug molecules.

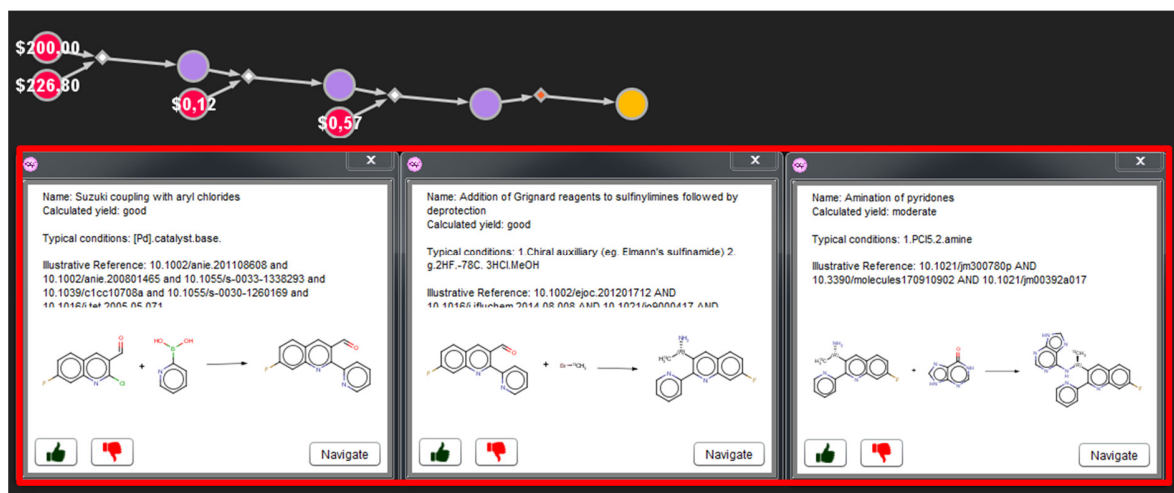


Figure S14. Details of the top-scoring pathway proposed by Chematica for the most accessible ^{13}C labeled M+1 isotopomer of AMG-319 (cf. main-text **Figure 10a**).

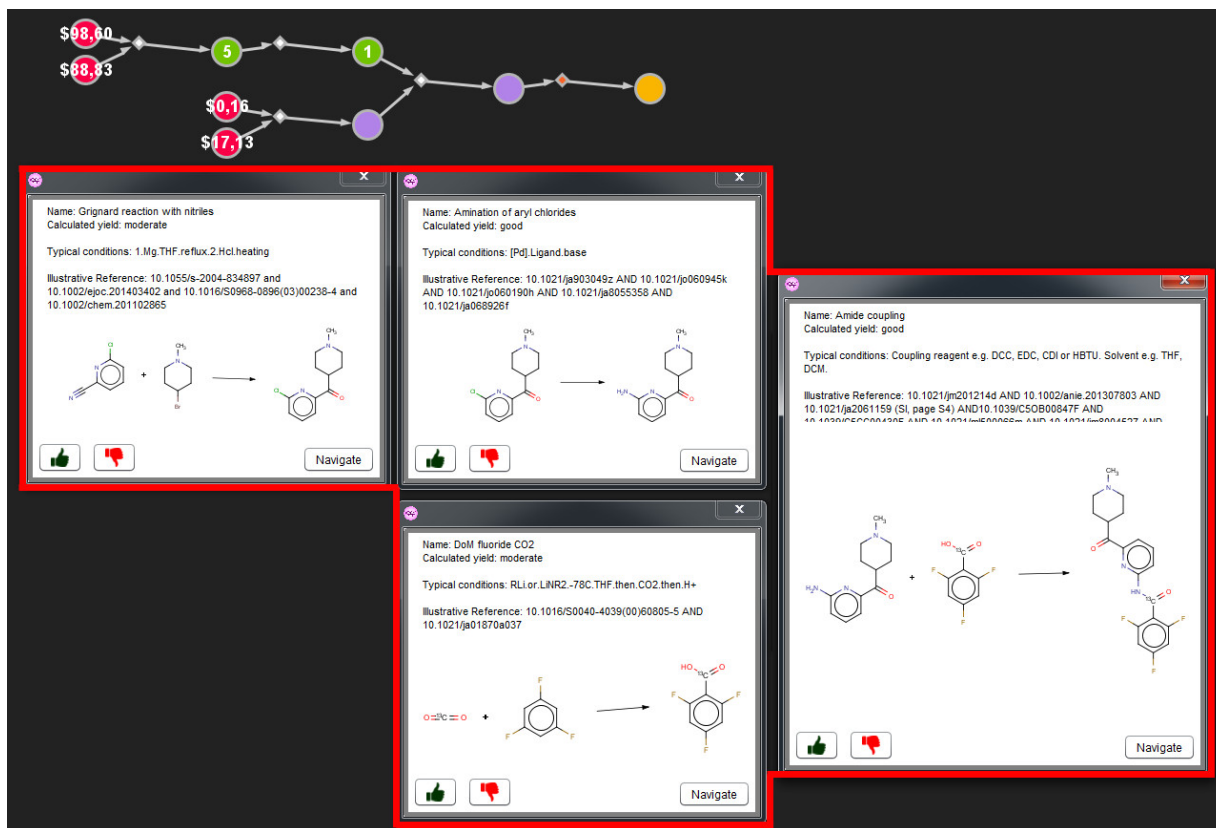


Figure S15. Details of the top-scoring pathway proposed by Chematica for the most accessible ^{13}C labeled M+1 isotopomer of Lasmiditan (cf. main-text **Figure 10b**).

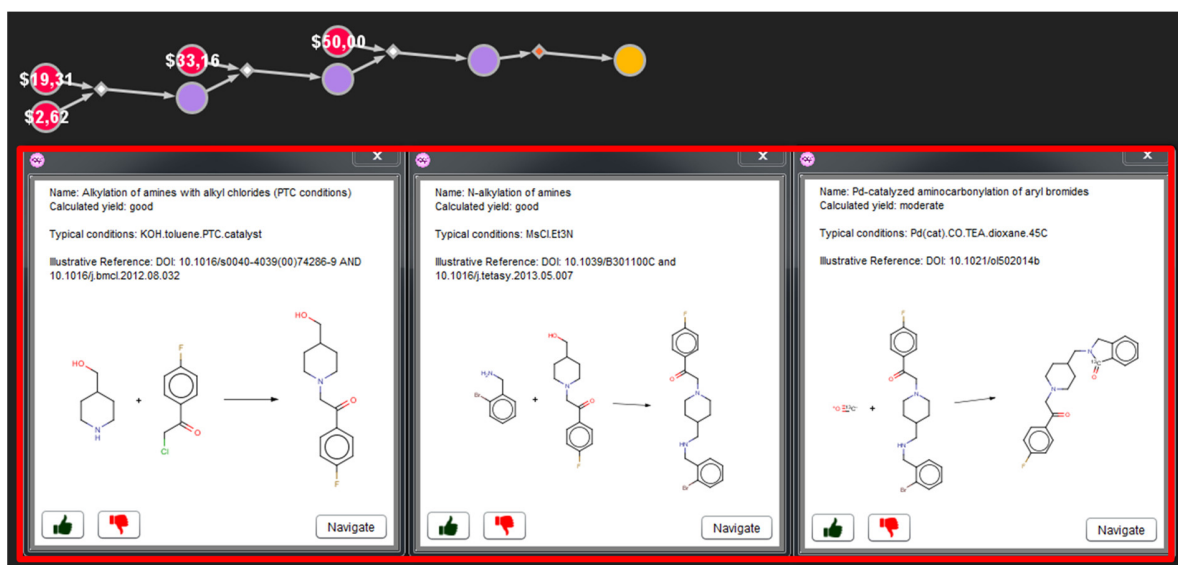


Figure S16. Details of the top-scoring pathway proposed by Chematica for the most accessible ^{13}C labeled M+1 isotopomer of Roluperidone (cf. main-text **Figure 10c**).

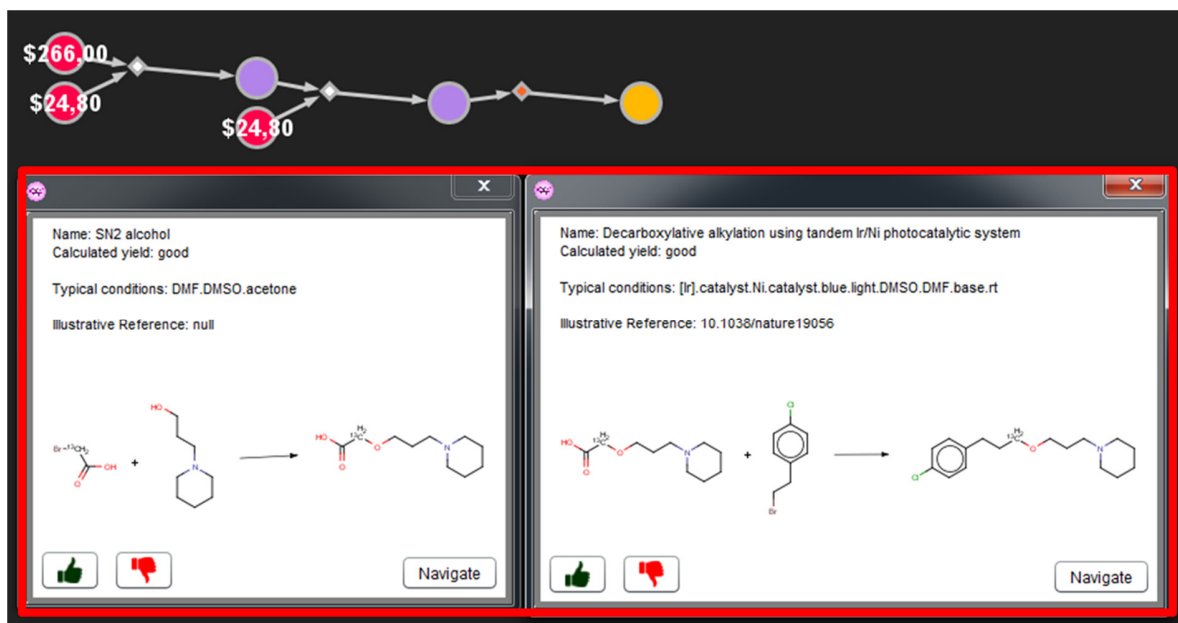


Figure S17. Details of the top-scoring pathway proposed by Chematica for the most accessible ^{13}C labeled M+1 isotopomer of Pitolisant (cf. main-text **Figure 10d**).

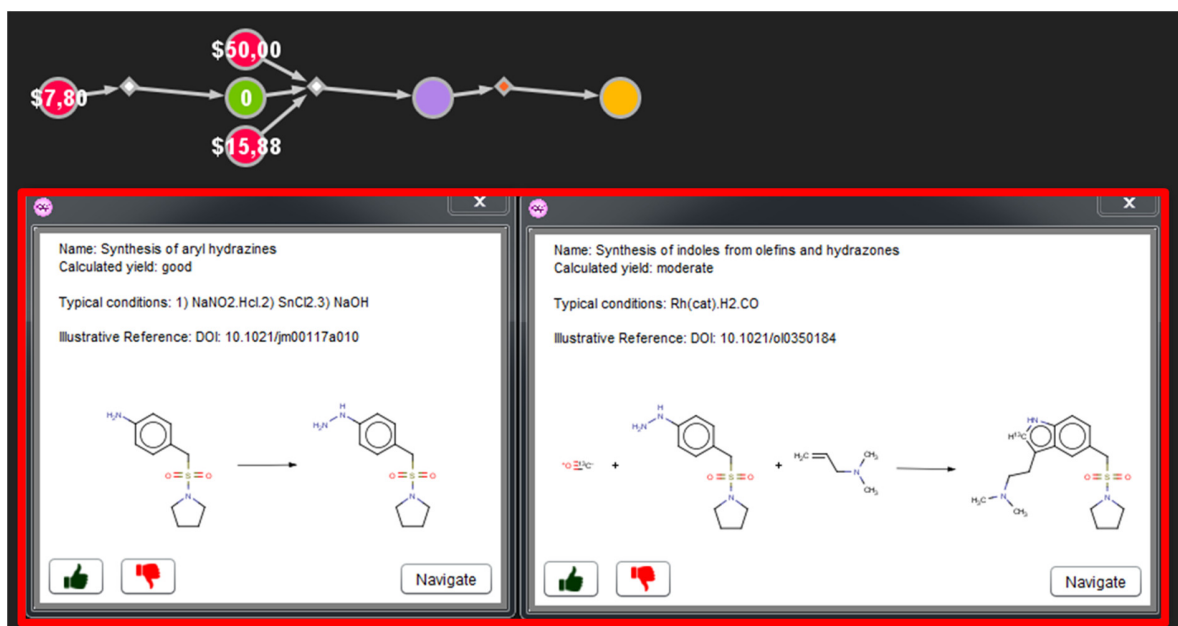


Figure S18. Details of the top-scoring pathway proposed by Chematica for the most accessible ¹³C labeled M+1 isotopomer of Almotriptan (cf. main-text **Figure 10e**).